- Q.1) Account (accno , coustomername , balance)

  a) Write trigger to generate primary key i.e. accno automatically. Account number will be like S00001, S00002,……,S00999 etc.

  PLSQL :

```
CREATE TABLE Account (
   accno VARCHAR(10) PRIMARY KEY,
   customername VARCHAR(255),
   balance DECIMAL(10, 2)
);

 CREATE OR REPLACE TRIGGER before_insert_Account
BEFORE INSERT ON Account
FOR EACH ROW
DECLARE
   new_accno VARCHAR2(10);
   max_accno INT;
BEGIN
   SELECT MAX(TO_NUMBER(SUBSTR(accno, 2))) INTO max_accno FROM
Account;

   new_accno := 'S' || LPAD(NVL(max_accno + 1, 1), 5, '0');

   :NEW.accno := new_accno;
END;
/
```

EDIT	ACCNO	CUSTOMERNAME	BALANCE
	S00001	Aniruddha	450.02
	S00002	Samim	154.25
	S00003	RUPKUMAR	500
			row(s) 1 - 3 of 3

b) Write down PL/SQL procedure which finds count of all even and odd balance records.

PLSQL:

```
CREATE OR REPLACE PROCEDURE FindEvenOddBalanceCount(
   evenCount OUT NUMBER,
```

```
    oddCount OUT NUMBER
)
AS
BEGIN
    -- Initialize counts to zero
    evenCount := 0;
    oddCount := 0;

    -- Iterate through records and count even and odd balances
    FOR rec IN (SELECT balance FROM Account)
    LOOP
        IF MOD(rec.balance, 2) = 0 THEN
            evenCount := evenCount + 1;
        ELSE
            oddCount := oddCount + 1;
        END IF;
    END LOOP;
END;
/

// procedure created
// To use the procedure //

DECLARE
    v_evenCount NUMBER;
    v_oddCount NUMBER;
BEGIN
    FindEvenOddBalanceCount(v_evenCount, v_oddCount);
    DBMS_OUTPUT.PUT_LINE('Even Balance Count: ' || v_evenCount);
    DBMS_OUTPUT.PUT_LINE('Odd Balance Count: ' || v_oddCount);
END;
/
```

Output

```
Even Balance Count: 1
Odd Balance Count: 2

Statement processed.
```

c) Write a trigger on each deposit or withdrawal on account such that a new record will be inserted in the table.
Transaction(Serialno , accno , transaction_type , transaction_date , oldbalance,newbalance)

Serialno should be generated automatically using a sequence. Transaction type is either debit or credit.

PLSQL:

// transaction table creation

```
CREATE TABLE Transaction (
   Serialno NUMBER PRIMARY KEY,
   accno VARCHAR2(10) REFERENCES Account(accno),
   transaction_type VARCHAR2(10),
   transaction_date DATE,
   oldbalance DECIMAL(10, 2),
   newbalance DECIMAL(10, 2)
);
```

//Trigger

- Question -2) Emp(empno , ename , job , mgr , hiredate , sal , comm , deptno) Dept(deptno , dname ,loc).

a) Write a PL/SQL cursor that will list the details of the employees whose salary in less than 3000 and works as a manager.

PLSQL:

```
DECLARE
  CURSOR emp_cursor IS
    SELECT * FROM emp
    WHERE sal < 3000 AND job = 'MANAGER';
BEGIN
  FOR emp_rec IN emp_cursor LOOP
    DBMS_OUTPUT.PUT_LINE('EmpNo: ' || emp_rec.empno || ', Ename: ' || emp_rec.ename
||
                         ', Job: ' || emp_rec.job || ', Salary: ' || emp_rec.sal);
  END LOOP;
END;
/
```

RESULTS :

```
EmpNo: 7698, Ename: BLAKE, Job: MANAGER, Salary: 2850
EmpNo: 7782, Ename: CLARK, Job: MANAGER, Salary: 2450
```

EmpNo: 7566, Ename: JONES, Job: MANAGER, Salary: 2975

Statement process

b) Write down a PL/SQL function or procedure that will take empno , jobs , as input and give a increment of 10 % if job in manager and 15 % else.

PLSQL:

```
create or replace PROCEDURE SALHIKE(eno IN NUMBER,jobrole IN
VARCHAR2) AS
BEGIN
IF jobrole = 'MANAGER' THEN
UPDATE EMP SET SAL=SAL*1.1 WHERE empno=eno;

ELSE
UPDATE EMP SET SAL=SAL*1.5 WHERE empno=eno;
END IF;
END SALHIKE;
```

RESULTS :

Procedure

c) Write trigger to generate primary key i.e. empno automatically. employee number will be like E00001, E00002,.....,E00999 etc.

PLSQL:

```
CREATE SEQUENCE empid_sequence
START WITH 1
INCREMENT BY 1
MAXVALUE 99999
NOCYCLE

CREATE OR REPLACE TRIGGER empid_sequence
BEFORE INSERT ON emp
FOR EACH ROW
BEGIN
:NEW.empno := 'E' || TO_CHAR(emp_sequence.NEXTVAL, 'FM00000');
```

```
END;
/
```
RESULTS:

- Q3)a> Account (accno , coustomername , balance) Minimum balance is 500
  a) Generate accno automatically with a trigger. Account number will be like S00001, S00002,, S00999 etc. It will also check validity of minimum balance.

PLSQL:

```
CREATE TABLE Account (
    accno VARCHAR2(10) PRIMARY KEY,
    customername VARCHAR2(50),
    balance NUMBER(10, 2) CHECK (balance >= 500)
);
CREATE OR REPLACE TRIGGER before_insert_Account
BEFORE INSERT ON Account
FOR EACH ROW
DECLARE
    new_accno VARCHAR2(10);
    max_accno INT;
BEGIN
    SELECT NVL(MAX(TO_NUMBER(SUBSTR(accno, 2))), 0) INTO max_accno FROM
Account;
    new_accno := 'S' || TO_CHAR(NVL(max_accno + 1, 1), 'FM00009');
    :NEW.accno := new_accno;
    IF :NEW.balance IS NULL THEN
        :NEW.balance := 0; -- Assuming a default value for balance if not provided
    END IF;
```

```
    IF :NEW.balance < 500 THEN
        RAISE_APPLICATION_ERROR(-20001, 'Minimum balance should be 500.');
    END IF;
END;
/
```

ACCNO	CUSTOMERNAME	BALANCE
S00001	Avirup	1000
S00002	Jana	700

b) Write PL/SQL code to count the prime and non-prime balance value for the account records.

PLSQL:

```
DECLARE
    prime_count NUMBER := 0;
    non_prime_count NUMBER := 0;
BEGIN
    FOR rec IN (SELECT * FROM Account) LOOP
        IF rec.balance IS NOT NULL THEN
            DECLARE
                i NUMBER;
                is_prime BOOLEAN := TRUE;
            BEGIN
                IF rec.balance <= 1 THEN
                    is_prime := FALSE;
                ELSE
                    FOR i IN 2..SQRT(rec.balance) LOOP
                        IF MOD(rec.balance, i) = 0 THEN
                            is_prime := FALSE;
                            EXIT;
                        END IF;
                    END LOOP;
                END IF;

                IF is_prime THEN
                    prime_count := prime_count + 1;
```

```
            ELSE
               non_prime_count := non_prime_count + 1;
            END IF;
         END;
      END IF;
   END LOOP;

   DBMS_OUTPUT.PUT_LINE('Prime Balance Count: ' || prime_count);
   DBMS_OUTPUT.PUT_LINE('Non-Prime Balance Count: ' || non_prime_count);
END;
/
```

Output based on the table value that used in 3.a table

```
Prime Balance Count: 0
Non-Prime Balance Count: 2
```

- 4) Message(msgno , msgtext ,mdate,from , to)

a) Generate msgid with a trigger, Message id will be like M0000001, M0000002 etc.

PLSQL:

```
CREATE TABLE messages (
   msgno VARCHAR2(10) PRIMARY KEY,
   msgtext VARCHAR2(4000),
   mdate DATE,
   sender VARCHAR2(100),
   receiver VARCHAR2(100)
);
CREATE OR REPLACE TRIGGER generate_msgid_trigger
BEFORE INSERT ON messages
FOR EACH ROW
DECLARE
   v_msgid VARCHAR2(10);
BEGIN
   -- Generate the msgid using the sequence
   SELECT 'M' || TO_CHAR(msgid_seq.NEXTVAL, 'FM0000000')
```

```
    INTO v_msgid
    FROM dual;

    -- Set the generated msgid for the new row
    :NEW.msgno := v_msgid;
END generate_msgid_trigger;

INSERT INTO messages (msgno, msgtext, mdate, sender, receiver)
VALUES (1, 'Hiiiii', SYSDATE, 'Gourab', 'Saikat');

INSERT INTO messages (msgno, msgtext, mdate, sender, receiver)
VALUES (2, 'Hello', SYSDATE, 'Saikat', 'Gourab');

select * from messages;
```

Results | Explain | Describe | Saved SQL | History

MSGNO	MSGTEXT	MDATE	SENDER	RECEIVER
M0000004	Hello	12/20/2023	Saikat	Gourab
M0000003	Hiiiii	12/20/2023	Gourab	Saikat

2 rows returned in 0.00 seconds Download

b) Write PL/SQL cursor that will insert all the messages whose length in less than 100 character in a seperate table. SMS(smsid , smstext, smsdate, from, to)

PLSQL:

```
CREATE TABLE SMS (
    smsid NUMBER PRIMARY KEY,
    smstext VARCHAR2(100),
    smsdate DATE,
    sender VARCHAR2(100),
```

```
    receiver VARCHAR2(100)
);


DECLARE
    CURSOR short_message_cursor IS
        SELECT msgno, msgtext, mdate, sender, receiver FROM messages WHERE
LENGTH(msgtext) < 100;
v_smsid SMS.smsid%TYPE;
    v_smstext SMS.smstext%TYPE;
    v_smsdate SMS.smsdate%TYPE;
    v_sender SMS.sender%TYPE;
    v_receiver SMS.receiver%TYPE;
BEGIN
    FOR message_rec IN short_message_cursor LOOP
        v_smsid := msgid_seq.NEXTVAL;
        v_smstext := message_rec.msgtext;
        v_smsdate := message_rec.mdate;
        v_sender := message_rec.sender;
        v_receiver := message_rec.receiver;
        INSERT INTO SMS(smsid, smstext, smsdate, sender, receiver)
        VALUES (v_smsid, v_smstext, v_smsdate, v_sender, v_receiver);
    END LOOP;
END;
SELECT * FROM sms;
```

Results | Explain | Describe | Saved SQL | History

SMSID	SMSTEXT	SMSDATE	SENDER	RECEIVER
5	Hello	12/20/2023	Saikat	Gourab
6	Hiiiii	12/20/2023	Gourab	Saikat

2 rows returned in 0.00 seconds Download

c) There is a daily message limit of 10 message per day. After that RS. 1/message will be charged. Implement this during messaging.

PLSQL:

```
CREATE OR REPLACE PROCEDURE send_message(
    p_msgtext IN VARCHAR2,
    p_mdate IN DATE,
    p_sender IN VARCHAR2,
    p_receiver IN VARCHAR2
) AS
    daily_limit CONSTANT NUMBER := 10;
    cost_per_message CONSTANT NUMBER := 1;
    remaining_messages NUMBER;
BEGIN
    SELECT COUNT(*)
    INTO remaining_messages
    FROM messages
    WHERE sender = p_sender
      AND TRUNC(mdate) = TRUNC(SYSDATE);
    IF remaining_messages >= daily_limit THEN
        INSERT INTO billing_history(sender, amount, transaction_date)
        VALUES (p_sender, cost_per_message, SYSDATE);
        DBMS_OUTPUT.PUT_LINE('Daily message limit exceeded. Rs. 1/message charged.');
    ELSE
        INSERT INTO messages(msgno, msgtext, mdate, sender, receiver)
        VALUES (NULL, p_msgtext, p_mdate, p_sender, p_receiver);
```

```
        DBMS_OUTPUT.PUT_LINE('Message sent successfully.');
    END IF;
END send_message;
```

- 5) Emp(empno , ename , job, mgr , hiredate , sal , comm , deptno)

a) Write down PL/SQL code that will display and update salary by 10% where employee has spent more 10 years in service.

PLSQL:

```
DECLARE
  CURSOR emp_cursor IS
    SELECT empno, ename, sal
    FROM emp
    WHERE MONTHS_BETWEEN(SYSDATE, hiredate) / 12 > 10;

BEGIN
  FOR emp_rec IN emp_cursor LOOP
    DBMS_OUTPUT.PUT_LINE('Employee: ' || emp_rec.empno || ' ' || emp_rec.ename || ',
Old Salary: ' || emp_rec.sal);
    UPDATE emp
    SET sal = sal * 1.1
    WHERE empno = emp_rec.empno;
    DBMS_OUTPUT.PUT_LINE('New Salary: ' || emp_rec.sal * 1.1);
  END LOOP;
END;
/
```

OUTPUT:

Results Explain Describe Saved SQL History

```
Employee: 7839 KING, Old Salary: 5000
New Salary: 5500
Employee: 7698 BLAKE, Old Salary: 2850
New Salary: 3135
Employee: 7782 CLARK, Old Salary: 2450
New Salary: 2695
Employee: 7566 JONES, Old Salary: 2975
New Salary: 3272.5
Employee: 7788 SCOTT, Old Salary: 3000
New Salary: 3300
Employee: 7902 FORD, Old Salary: 3000
New Salary: 3300
Employee: 7369 SMITH, Old Salary: 800
New Salary: 880
Employee: 7499 ALLEN, Old Salary: 1600
New Salary: 1760
Employee: 7521 WARD, Old Salary: 1250
New Salary: 1375
Employee: 7654 MARTIN, Old Salary: 1250
New Salary: 1375
Employee: 7844 TURNER, Old Salary: 1500
New Salary: 1650
Employee: 7876 ADAMS, Old Salary: 1100
New Salary: 1210
Employee: 7900 JAMES, Old Salary: 950
New Salary: 1045
Employee: 7934 MILLER, Old Salary: 1300
New Salary: 1430

1 row(s) updated.
```

0.01 seconds

b) Write down trigger that will store all salary updates on employees in separate table with necessary data.

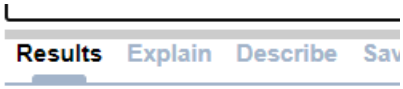
PLSQL:

```
CREATE TABLE salary_update_log(
empno NUMBER,
old_sal NUMBER,
new_sal NUMBER,
update_date DATE
);
```

```
CREATE OR REPLACE TRIGGER salary_update_trigger

AFTER UPDATE OF sal ON Emp

FOR EACH ROW

BEGIN
```

```
  INSERT INTO salary_update_log (empno, old_sal, new_sal, update_date)
  VALUES (:old.empno, :old.sal, :new.sal, SYSDATE);
END;
```

/Output:




```
Trigger created.

0.01 seconds
```

c) Find Junior most employee in each dept and give a 10% incentive.

PLSQL:

```
DECLARE
min_hire_date DATE;
emp_recemp%ROWTYPE;
  CURSOR dept_cur IS
    SELECT deptno
    FROM emp
    GROUP BY deptno;
dept_recdept_cur%ROWTYPE;
BEGIN
  OPEN dept_cur;
  LOOP
    FETCH dept_cur INTO dept_rec;
    EXIT WHEN dept_cur%NOTFOUND;

    SELECT hiredate INTO min_hire_date
    FROM emp
```

```
    WHERE deptno = dept_rec.deptno
    ORDER BY hiredate DESC
    FETCH FIRST 1 ROW ONLY;

    SELECT * INTO emp_rec
    FROM emp
    WHERE deptno = dept_rec.deptno
    AND hiredate = min_hire_date;

    UPDATE emp SET sal = sal * 1.1 WHERE empno = emp_rec.empno;
    DBMS_OUTPUT.PUT_LINE('Junior employee in dept ' || dept_rec.deptno || ' is ' ||
emp_rec.ename);
  END LOOP;
  CLOSE dept_cur;
END;
/
```

➢ 6) A student table contains student records like
Student (Roll, Name, Semester, Paper, Year, Marks, Grade)

PLSQL:

```
CREATE TABLE Student (
    Roll VARCHAR2(10),
    Name VARCHAR2(50),
    Semester VARCHAR2(10),
    Paper VARCHAR2(50),
    Year NUMBER,
    Marks NUMBER,
    Grade VARCHAR2(2)
);
```

a) Generate Roll number with a trigger. Roll no will be like S00001, S00002,…..,

S00999 etc
PLSQL:

```
CREATE SEQUENCE roll_sequence
   START WITH 1
   INCREMENT BY 1
   MAXVALUE 999
   CYCLE
   NOCACHE;

CREATE OR REPLACE TRIGGER student_roll_trigger
BEFORE INSERT ON Student
FOR EACH ROW
DECLARE
v_sequence_number NUMBER;
BEGIN
   SELECT roll_sequence.NEXTVAL INTO v_sequence_number FROM dual;
:NEW.Roll := 'S' || TO_CHAR(v_sequence_number, 'FM00000');
END;
/
```

b) Write PL/SQL code to find grade of the students based on marks.
PLSQL:

```
DECLARE
v_marksNUMBER := 85; -- Replace with the actual marks
v_grade VARCHAR2(2);
BEGIN
   IF v_marks>= 90 THEN
v_grade := 'A';
   ELSIF v_marks>= 80 THEN
v_grade := 'B';
   ELSIF v_marks>= 70 THEN
v_grade := 'C';
   ELSIF v_marks>= 60 THEN
v_grade := 'D';
   ELSE
v_grade := 'F';
   END IF;

   DBMS_OUTPUT.PUT_LINE('Grade: ' || v_grade);
END;
```

/

c) Write trigger before on Student such that a failed student record will be inserted in a separate table. Marks will be inserted if it is lesser than full marks. If not give appropriate error message.
ArrearMaster (Slno, Roll, Semester, Paper, Year, Marks).

PLSQL:

```
CREATE TABLE ArrearMaster (
Slno NUMBER,
   Roll VARCHAR2(10),
   Semester VARCHAR2(10),
   Paper VARCHAR2(50),
   Year NUMBER,
   Marks NUMBER
);

CREATE OR REPLACE TRIGGER student_failed_trigger
BEFORE INSERT ON Student
FOR EACH ROW
DECLARE
v_full_marksNUMBER := 100; -- Replace with the full marks for the paper
BEGIN
IF :NEW.Marks<v_full_marks THEN
      INSERT INTO ArrearMaster (Slno, Roll, Semester, Paper, Year, Marks)
      VALUES
(ArrearMaster_Seq.NEXTVAL, :NEW.Roll, :NEW.Semester, :NEW.Paper, :NEW.Year, :NEW
.Marks);
   ELSE
      RAISE_APPLICATION_ERROR(-20001, 'Error: Marks cannot be greater than or
equal to full marks.');
   END IF;
END;
/
```

- 8) General PL/SQL procedure/function

1:HCF of two numbers

```
DECLARE

        num1 INTEGER;
```

```
        num2 INTEGER;
        t INTEGER;
BEGIN
        num1 := 8;

        num2 := 48;

        WHILE MOD(num2, num1) != 0 LOOP
                t := MOD(num2, num1);

                num2 := num1;

                num1 := t;
        END LOOP;

        dbms_output.Put_line('HCF of '||num1 ||' and '||num2 ||' is '||num1);
END;

O/P: HCF of 8 and 48 is 8

Statement processed.
2:checkingarmstrong number
declare

        n number:=1634;
        s number:=0;
        r number;
        len number;
        m number;

begin
        m := n;

        len := length(to_char(n));

        -- while loop till n>0
        while n>0
        loop
                r := mod(n , 10);
                s := s + power(r , len);
                n := trunc(n / 10);
        end loop;

        if m = s
        then
```

```
                dbms_output.put_line('yes');
        else
                dbms_output.put_line('no');
        end if;

end;
```

o/p: yes

Statement processed.

3:prime factor of a number

```
declare

n number;
i number;
temp number;

begin


n := 13;


i := 2;


temp := 1;


for i in 2..n/2
        loop
                if mod(n, i) = 0
                then
                        temp := 0;
                        exit;
                end if;
        end loop;

        if temp = 1
        then
                dbms_output.put_line('true');
        else
                dbms_output.put_line('false');
        end if;
end;
```

o/p:true
4:Fibonacci series

```
declare
first number := 0;
second number := 1;
temp number;

n number := 5;
i number;

begin

        dbms_output.put_line('Series:');


        dbms_output.put_line(first);
        dbms_output.put_line(second);


        for i in 2..n
        loop
                temp:=first+second;

first := second;
second := temp;
        dbms_output.put_line(temp);
end loop;

end;
```

o/p:

```
Series:
0
1
1
2
3
5
```

5:name abbreviation

```
DECLARE
first_name VARCHAR2(50) := 'Samim';
last_name VARCHAR2(50) := 'Piyada';

   abbreviation VARCHAR2(10);
```

```
BEGIN
abbreviation := SUBSTR(first_name, 1, 1);

abbreviation := abbreviation || '.';

abbreviation := abbreviation || SUBSTR(last_name, 1, 1);

   DBMS_OUTPUT.PUT_LINE('Abbreviation: ' || abbreviation);
END;
/
o/p:
Abbreviation: S.P

Statement processed.
6.Binary search
CREATE OR REPLACE FUNCTION binary_search(
arr IN SYS.ODCINumberList,
   target IN NUMBER
) RETURN NUMBER
IS
   low PLS_INTEGER := arr.FIRST;
   high PLS_INTEGER := arr.LAST;
   mid PLS_INTEGER;
BEGIN
   WHILE low <= high LOOP
mid := (low + high) / 2;

      IF arr(mid) = target THEN
         RETURN mid; -- Element found, return its index
      ELSIF arr(mid) < target THEN
         low := mid + 1; -- Adjust the search range
      ELSE
high := mid - 1;
      END IF;
   END LOOP;

   RETURN -1;
END binary_search;
/
o/p:Function created
DECLARE
   numbers SYS.ODCINumberList := SYS.ODCINumberList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);

   target NUMBER := 7;
```

```
   result NUMBER;
BEGIN
result := binary_search(numbers, target);

   IF result >= 0 THEN
      DBMS_OUTPUT.PUT_LINE('Element ' || target || ' found at index ' || result);
   ELSE
      DBMS_OUTPUT.PUT_LINE('Element ' || target || ' not found in the array');
   END IF;
END;
/
```

o/p:

```
Element 7 found at index 7

Statement processed.
```

7.Linear search

```
CREATE OR REPLACE FUNCTION linear_search(
arr IN SYS.ODCINumberList,
   target IN NUMBER
) RETURN NUMBER
IS
result_index PLS_INTEGER := -1;
BEGIN
   FOR i IN arr.FIRST .. arr.LAST LOOP
      IF arr(i) = target THEN
result_index := i;
         EXIT;
      END IF;
   END LOOP;

   RETURN result_index;
END linear_search;
/
```

o/p:Function created

```
DECLARE

   numbers SYS.ODCINumberList := SYS.ODCINumberList(10, 20, 30, 40, 50, 60, 70, 80,
90, 100);


   target NUMBER := 60;


   result NUMBER;
```

```
BEGIN

result := linear_search(numbers, target);


   IF result >= 0 THEN
      DBMS_OUTPUT.PUT_LINE('Element ' || target || ' found at index ' || result);
   ELSE
      DBMS_OUTPUT.PUT_LINE('Element ' || target || ' not found in the array');
   END IF;
END;
/
o/p:Element 60 found at index 6

Statement processed.

8.Bubble sort


CREATE OR REPLACE PROCEDURE bubble_sort(
arr IN OUT SYS.ODCINumberList
)
IS
   n PLS_INTEGER := arr.COUNT;
i PLS_INTEGER;
   j PLS_INTEGER;
   temp NUMBER;
BEGIN
   FOR i IN 1..n-1 LOOP
      FOR j IN 1..n-i LOOP
         -- Swap elements if they are in the wrong order
         IF arr(j) >arr(j+1) THEN
temp := arr(j);
arr(j) := arr(j+1);
arr(j+1) := temp;
         END IF;
      END LOOP;
   END LOOP;
```

```
END bubble_sort;
/
```

o/p:Prosedure created

```
   DECLARE
   numbers SYS.ODCINumberList := SYS.ODCINumberList(64, 25, 12, 22, 11, 1, 3, 120);

   PROCEDURE display_array(arr IN SYS.ODCINumberList) IS
   BEGIN
      FOR i IN arr.FIRST .. arr.LAST LOOP
DBMS_OUTPUT.PUT(arr(i) || ' ');
      END LOOP;
      DBMS_OUTPUT.NEW_LINE;
   END display_array;

BEGIN
   DBMS_OUTPUT.PUT_LINE('Unsorted array:');
display_array(numbers);

bubble_sort(numbers);

   DBMS_OUTPUT.PUT_LINE('Sorted array:');
display_array(numbers);
END;
/
```

o/p:
Unsorted array:
64 25 12 22 11 1 3 120
Sorted array:
1 3 11 12 22 25 64 120

Statement processed.

9.Grade Calculation

```
CREATE OR REPLACE FUNCTION calculate_grade(
in_score IN NUMBER
) RETURN VARCHAR2
IS
   grade VARCHAR2(2);
BEGIN
   IF in_score>= 90 THEN
grade := 'A';
   ELSIF in_score>= 80 THEN
```

```
grade := 'B';
   ELSIF in_score>= 70 THEN
grade := 'C';
   ELSIF in_score>= 60 THEN
grade := 'D';
   ELSE
grade := 'F';
   END IF;

   RETURN grade;
END calculate_grade;
/
DECLARE
student_scoreNUMBER := 75;
result_grade VARCHAR2(2);
BEGIN
result_grade := calculate_grade(student_score);
   DBMS_OUTPUT.PUT_LINE('Student Score: ' || student_score);
   DBMS_OUTPUT.PUT_LINE('Grade: ' || result_grade);
END;
/
```

o/p:Student Score: 75
Grade: C

Statement processed.

10.student record insertions using record type

PLSQL:

```
CREATE OR REPLACE TYPE student_record_type AS OBJECT (
student_id NUMBER,
student_name VARCHAR2(50),
student_grade VARCHAR2(2)
);
/

-- Create a table to store student records
CREATE TABLE student_table (
student_infostudent_record_type
);
/

-- Procedure to insert a student record
CREATE OR REPLACE PROCEDURE insert_student_record(
```

```
in_student_id NUMBER,
in_student_name VARCHAR2,
in_student_grade VARCHAR2
)
IS
student_info_varstudent_record_type;
BEGIN
student_info_var := student_record_type(in_student_id, in_student_name,
in_student_grade);

   INSERT INTO student_table VALUES (student_info_var);

   DBMS_OUTPUT.PUT_LINE('Student record inserted successfully.');
END insert_student_record;
/
```

o/p:

Type created.

- Q9>Write a procedure with no parameters. The procedure should say whether the current day is a weekend or weekday. Additionally, it should tell you the user's name and the current time. It also should specify how many valid and invalid procedures are in the database.

PLSQL:

```
CREATE OR REPLACE PROCEDURE MYProcedure IS
  v_current_date DATE;
  v_day_number NUMBER;
  v_username VARCHAR2(30);
  vcnt NUMBER;
  ivcnt NUMBER;
BEGIN


  v_current_date := SYSDATE;
  DBMS_OUTPUT.PUT_LINE('Current Date: ' || TO_CHAR(v_current_date, 'DD-MON-YYYY
HH24:MI:SS'));
  v_day_number := TO_NUMBER(TO_CHAR(v_current_date, 'D'));
```

```
  IF v_day_number IN (1, 7) THEN
    DBMS_OUTPUT.PUT_LINE('It is the weekend.');
  ELSE
    DBMS_OUTPUT.PUT_LINE('It is a weekday.');
  END IF;
v_username := USER;
DBMS_OUTPUT.PUT_LINE('The Database User Name = ' || v_username);

SELECT COUNT(*) INTO vcnt
  FROM ALL_OBJECTS
  WHERE OBJECT_TYPE = 'PROCEDURE' AND STATUS = 'VALID' ;

SELECT COUNT(*) INTO ivcnt
  FROM ALL_OBJECTS
  WHERE OBJECT_TYPE = 'PROCEDURE' AND STATUS = 'INVALID';

  DBMS_OUTPUT.PUT_LINE('Valid Procedure Count: ' || vcnt);
  DBMS_OUTPUT.PUT_LINE('Invalid Procedure Count: ' || ivcnt);

END MYProcedure;
/
BEGIN
  MYProcedure;
END;
/
```

```
  DBMS_OUTPUT.PUT_LINE('Valid Procedure Count: ' || vcnt);
  DBMS_OUTPUT.PUT_LINE('Invalid Procedure Count: ' || ivcnt);
END MYProcedure;
/

BEGIN
  MYProcedure;
END;
/
```

Results Explain Describe Saved SQL History

```
Current Date: 20-DEC-2023 19:55:33
It is a weekday.
The Database User Name = ANONYMOUS
Valid Procedure Count: 36
Invalid Procedure Count: 1

Statement processed.

0.02 seconds
```

- Q. 10 Write a procedure that takes in a zip code, city, and state and inserts the values into the zip codetable. It should check to see if the zip code is already in the database. If it is, an exception shouldbe raised, and an error message should be displayed. Write an anonymous block that uses the procedure and inserts your zip code

- Step 1: 1st create a table using this:

```
CREATE TABLE zip_codetable (
zip_code VARCHAR(10) PRIMARY KEY,
city VARCHAR(50),
state VARCHAR(2)
);
```

- Step 2: insert values into the table:

```
INSERT INTO zip_codetable (zip_code, city, state)
VALUES
 ('12345', 'Sample City 1', 'CA'),
```

```
  ('67890', 'Sample City 2', 'NY'),
  ('11111', 'Sample City 3', 'TX');
```

ZIP_CODE	CITY	STATE
11111	Sample City 3	TX
12345	Sample City 1	CA
67890	Sample City 2	NY

Procedure :

```
CREATE OR REPLACE PROCEDURE insert_zip_code(
  p_zip_code IN VARCHAR2,
  p_city IN VARCHAR2,
  p_state IN VARCHAR2
) AS
  v_count NUMBER;

BEGIN
  SELECT COUNT(*) INTO v_count
  FROM zip_codetable
  WHERE zip_code = p_zip_code;

  IF v_count > 0 THEN
    RAISE_APPLICATION_ERROR(-20001, 'Zip code already exists in the database');
  END IF;
  INSERT INTO zip_codetable(zip_code, city, state)
  VALUES (p_zip_code, p_city, p_state);

  COMMIT;
END insert_zip_code;
/
```

❖ Calling the Procedure

```
DECLARE
  v_zip_code VARCHAR2(10) := '12345';
  v_city VARCHAR2(50) := 'Sample City';
  v_state VARCHAR2(2) := 'CA';
BEGIN
  insert_zip_code(v_zip_code, v_city, v_state);
  DBMS_OUTPUT.PUT_LINE('Zip code inserted successfully');
EXCEPTION
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
END;
/
```

It's showing error because the zip code already present in our table that we created previously .

```
Error: ORA-20001: Zip code already exists in the database

Statement processed.

0.00 seconds
```

1.HCF of two numbers

```
-- Package Specification
CREATE OR REPLACE PACKAGE MathPackage AS
  FUNCTION findHCF(num1 IN INTEGER, num2 IN INTEGER) RETURN INTEGER;
END MathPackage;
/


-- Package Body
CREATE OR REPLACE PACKAGE BODY MathPackage AS
  FUNCTION findHCF(num1 IN INTEGER, num2 IN INTEGER) RETURN INTEGER IS
    t INTEGER;
```

```
  BEGIN
    WHILE MOD(num2, num1) != 0 LOOP
t := MOD(num2, num1);
      num2 := num1;
      num1 := t;
    END LOOP;
    RETURN num1;
  END findHCF;
END MathPackage;
/
Plsql:
DECLARE
  result INTEGER;
BEGIN
result := MathPackage.findHCF(8, 48);
dbms_output.Put_line('HCF is ' || result);
END;
/
2.checking armstrong number
CREATE OR REPLACE PACKAGE ArmstrongPackage AS
    FUNCTION isArmstrongNumber(n IN NUMBER) RETURN VARCHAR2;
END ArmstrongPackage;
/

CREATE OR REPLACE PACKAGE BODY ArmstrongPackage AS
    FUNCTION isArmstrongNumber(n IN NUMBER) RETURN VARCHAR2 IS
        s NUMBER := 0;
```

```
        r NUMBER;
len NUMBER;
        m NUMBER;
    BEGIN
m := n;
len := LENGTH(TO_CHAR(n));

        -- while loop till n>0
        WHILE n > 0 LOOP
r := MOD(n, 10);
s := s + POWER(r, len);
n := TRUNC(n / 10);
        END LOOP;

        IF m = s THEN
            RETURN 'yes';
        ELSE
            RETURN 'no';
        END IF;
    END isArmstrongNumber;
END ArmstrongPackage;
/
Plsql
DECLARE
    result VARCHAR2(3);
    n NUMBER := 1634; -- Replace with the number you want to check
BEGIN
```

```
result := ArmstrongPackage.isArmstrongNumber(n);

   DBMS_OUTPUT.PUT_LINE(result);

END;

/
```

3.prime factor of a number

```
-- Create a package specification
CREATE OR REPLACE PACKAGE PrimePackage AS
  -- Function to check if a number is prime
  FUNCTION isPrime(n IN NUMBER) RETURN BOOLEAN;
END PrimePackage;
/

-- Create a package body
CREATE OR REPLACE PACKAGE BODY PrimePackage AS
  -- Function to check if a number is prime
  FUNCTION isPrime(n IN NUMBER) RETURN BOOLEAN IS
i NUMBER;
    temp NUMBER := 1;
  BEGIN
    -- Check if n is less than 2
    IF n < 2 THEN
      RETURN FALSE;
    END IF;

    -- Loop to check for factors
    FOR i IN 2..n/2 LOOP
      IF MOD(n, i) = 0 THEN
temp := 0;
        EXIT;
      END IF;
    END LOOP;

    -- Check the value of temp and return the result
    IF temp = 1 THEN
      RETURN TRUE;
    ELSE
      RETURN FALSE;
    END IF;
  END isPrime;
END PrimePackage;
/
```

4.Fibonacci series

```
-CREATE OR REPLACE PACKAGE FibonacciPackage AS
  PROCEDURE GenerateFibonacci(n IN NUMBER);
END FibonacciPackage;
/

CREATE OR REPLACE PACKAGE BODY FibonacciPackage AS
  PROCEDURE GenerateFibonacci(n IN NUMBER) IS
    first NUMBER := 0;
    second NUMBER := 1;
    temp NUMBER;

i NUMBER;

  BEGIN
    DBMS_OUTPUT.PUT_LINE('Fibonacci Series:');

    DBMS_OUTPUT.PUT_LINE(first);
    DBMS_OUTPUT.PUT_LINE(second);

    FOR i IN 2..n LOOP
temp := first + second;
first := second;
second := temp;
      DBMS_OUTPUT.PUT_LINE(temp);
    END LOOP;
  END GenerateFibonacci;
```

```
END FibonacciPackage;
/
sql
DECLARE
  n NUMBER := 5;
BEGIN
FibonacciPackage.GenerateFibonacci(n);
END;
/
```

5.name abbreviation

```
-- Create a package specification
CREATE OR REPLACE PACKAGE ArmstrongPackage AS
    -- Function to check if a number is an Armstrong number
    FUNCTION isArmstrongNumber(n NUMBER) RETURN BOOLEAN;
END ArmstrongPackage;
/

-- Create a package body
CREATE OR REPLACE PACKAGE BODY ArmstrongPackage AS
    -- Function implementation
    FUNCTION isArmstrongNumber(n NUMBER) RETURN BOOLEAN IS
        s NUMBER := 0;
        r NUMBER;
len NUMBER;
        m NUMBER := n;
    BEGIN
```

```
len := LENGTH(TO_CHAR(n));

        -- while loop till n>0
        WHILE n > 0 LOOP
r := MOD(n, 10);
s := s + POWER(r, len);
n := TRUNC(n / 10);
        END LOOP;

        -- Check if it's an Armstrong number
        RETURN m = s;
    END isArmstrongNumber;
END ArmstrongPackage;
/
```